Maze (Scratch)

Can you make it to the end?

Teachers' guide

An activity by the Australian Computing Academy

Teacher's notes

Have you ever designed a maze?

Humans have been designing mazes and labyrinths for over 2,500 years. They can be designed to amuse, confuse, and conceal.

Did you know: a maze is made up of branching paths, while a labyrinth is a single path going through a number of twists and turns.

Teacher's notes

What we'll be creating

In this activity students will:

- Familiarise themselves with the Scratch programming environment
- Create a sequence of instructions for a sprite
- Learn to use the keyboard as an input, to control the movement of a sprite
- Use conditional statements to determine whether the sprite can proceed or not (to prevent the sprite moving through the maze walls)
- Differentiated learning: students will also have an opportunity to explore the use of:
    - variables, to create points or lives in their game
    - loops, to control the appearance of the heart sprite
    - Extra backdrops, to create extra levels in the maze

The nuts and bolts

Suggested year groups: **Years 3 to 6**

Subject areas: **Digital technologies**

Suggested timing: **3 to 4 hours**

This guide is designed for use by teachers.
Click here for the accompanying student handout or download it at cmp.ac/DTMazeStudent.

Set up

Set up steps

- Look at the finished maze project here (https://scratch.mit.edu/projects/238494728/)

You will need:

- Access to Scratch: either at www.scratch.mit.edu or an offline version
- For students working online, student accounts. Students can create their own account (using an email address) or teachers can create a teacher verified account and set up a class of students.

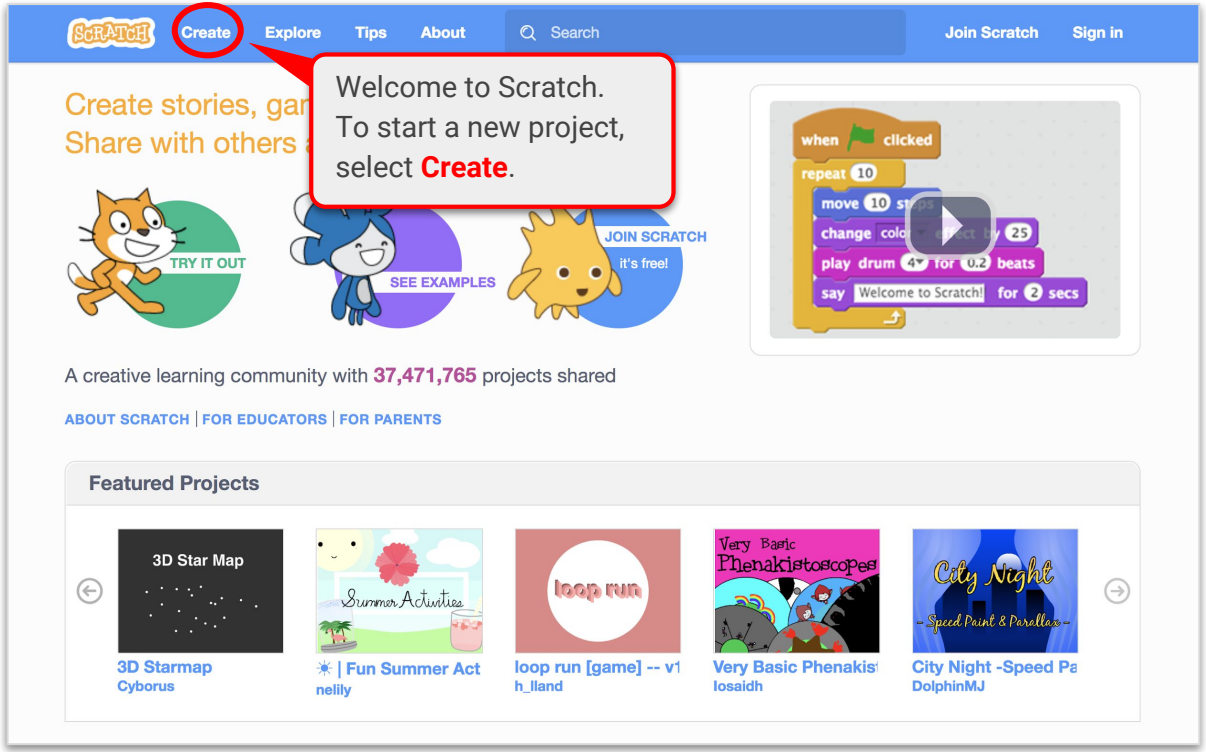
If you are just starting with Scratch, there are a number of free tutorials available on www.scratch.mit.edu to help you.

These slides are based on Scratch 3 which you can preview now at preview.scratch.mit.edu.

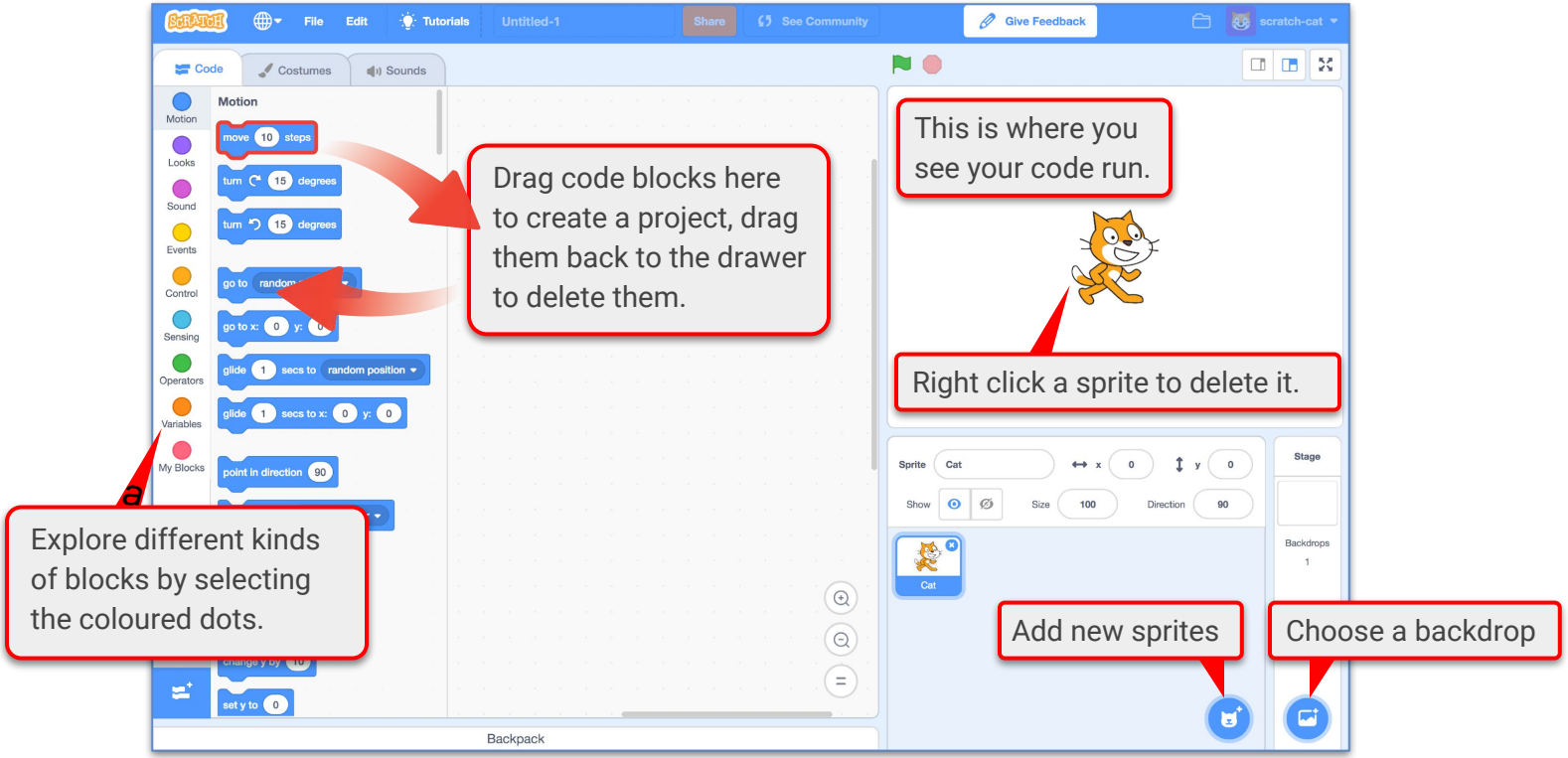
You can also use Scratch 2 or the offline version of Scratch to make this project.

Steps

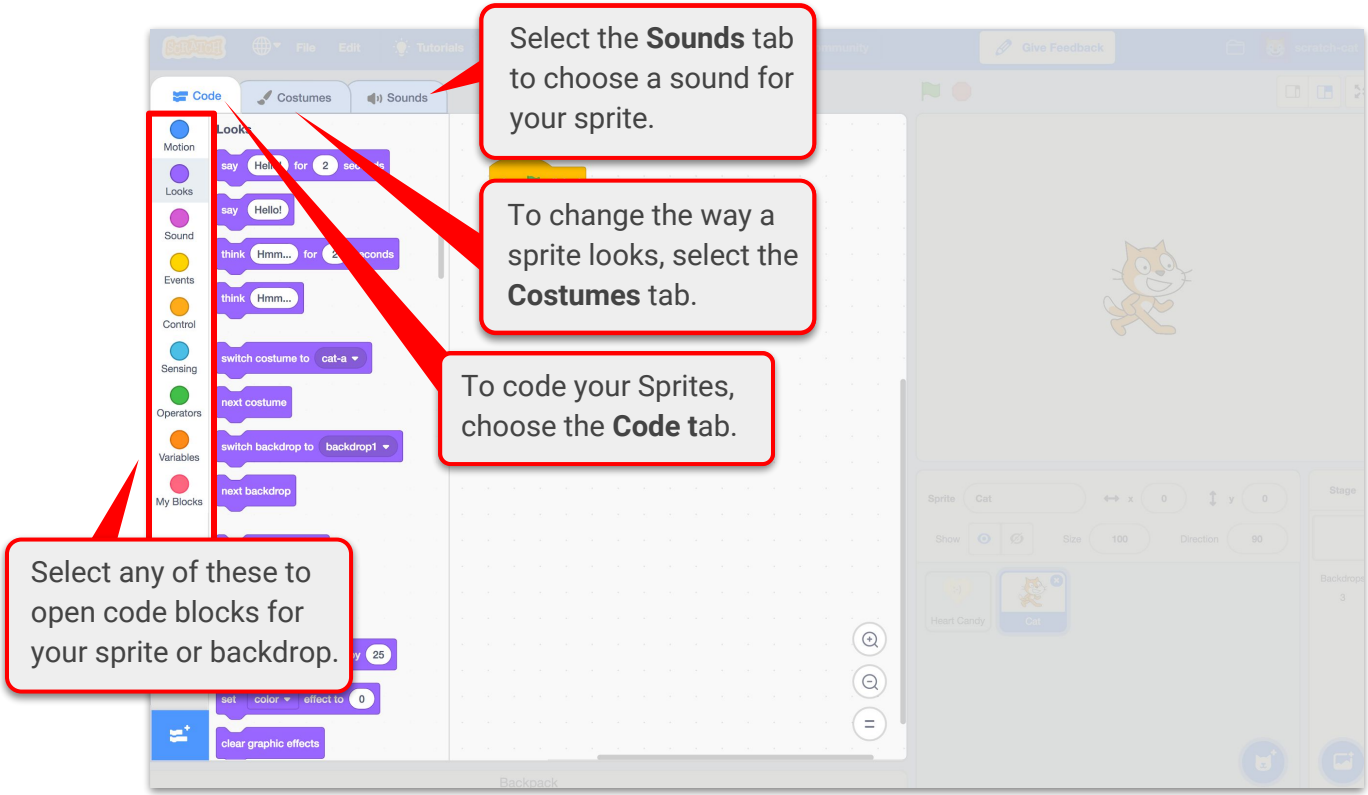
Step 1: Getting Started with Scratch



Step 1 (continued): Creating your first project

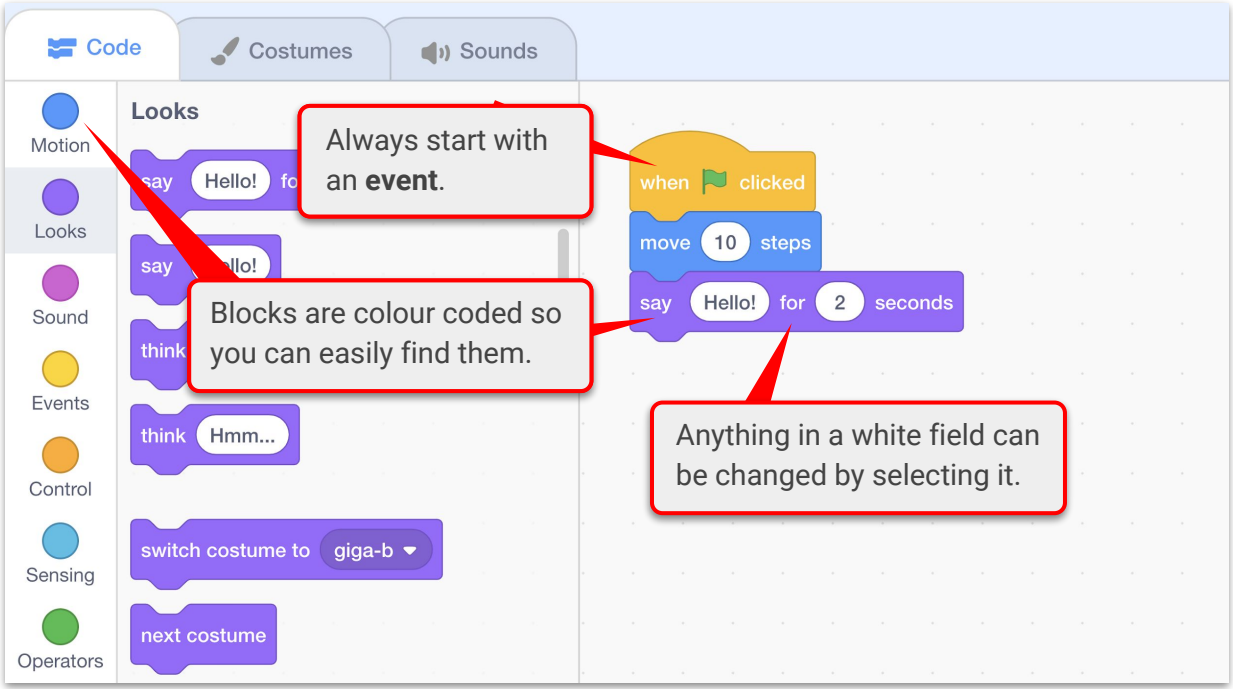


Step 1 (continued): Working with sprites

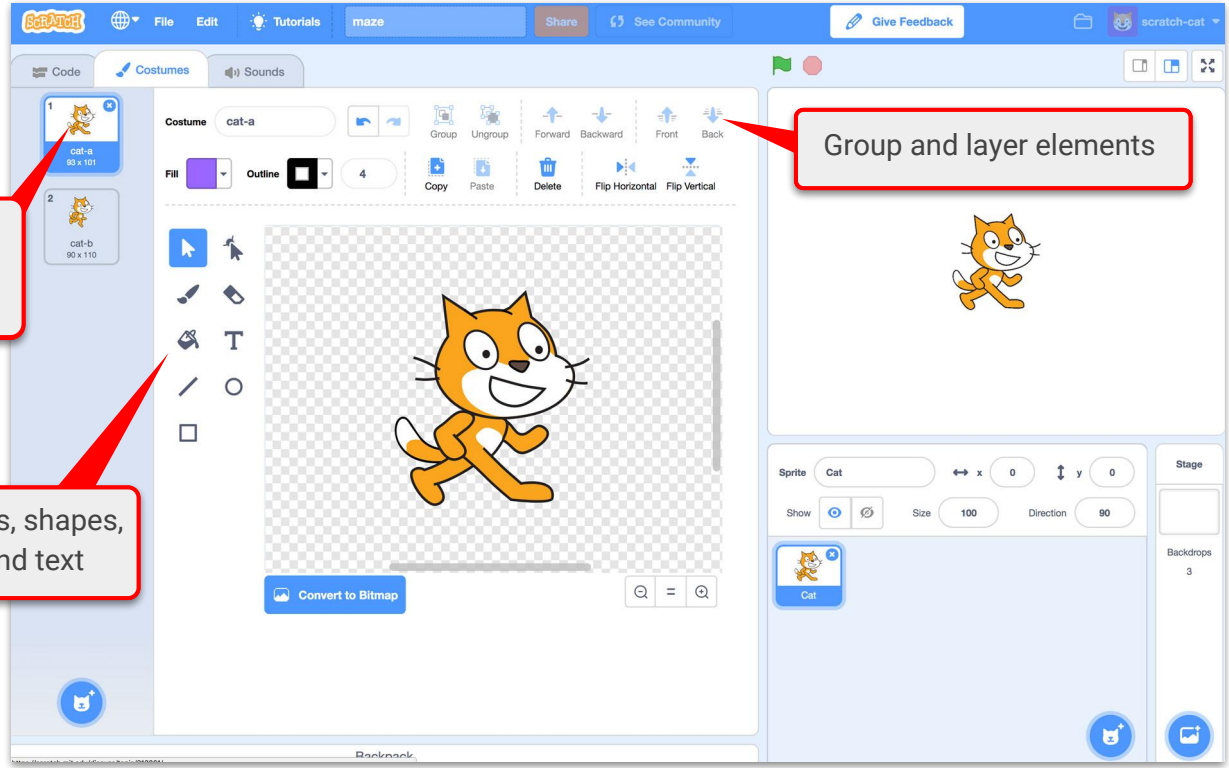


Step 1 (continued): Adding some code

Drag out these blocks to see what will happen.



Step 1 (continued): Designing sprites



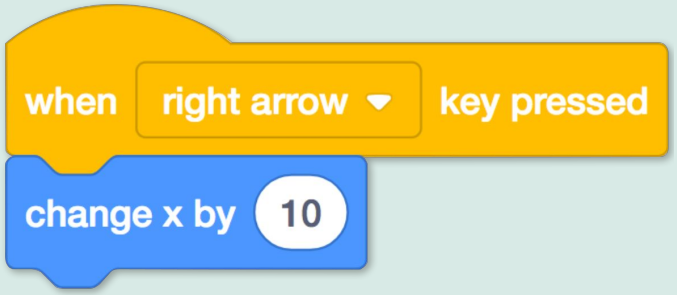
Step 2: What is a maze? Discussion activity

- Take a look at the sample project (https://scratch.mit.edu/projects/238494728/) we'll be building.
- **What elements do you need to make a working maze game?** Students work to create a list that may include some or all of the following:
  - A character to navigate through the maze
  - A maze - in Scratch represented as a birds-eye 2 dimensional view with obstacles to prevent the sprite travelling from start to end without avoiding obstacles
  - A start point
  - An end point
  - Points or tokens to collect along the way
  - Additional mazes to play through once the first level is complete
  - Obstacles to avoid (possibly moving)
- Play the game as a group and ask the following:
  - **How is the sprite controlled?** (With the up, down, left and right arrow keys)
  - **How else could you control it ?** (Other keyboard options include WASD keys, or if time permits this project works well in combination with a makey makey set or with the micro:bit integration in Scratch 3)
  - **What happens when the sprite tries to go through a wall?**
  - **How does the player know they have reached the end of the maze?**

Step 3: Investigating the code

Now that students have played the Scratch maze project, investigate the code blocks inside the project. Ask the following questions:

Take a look at this code.

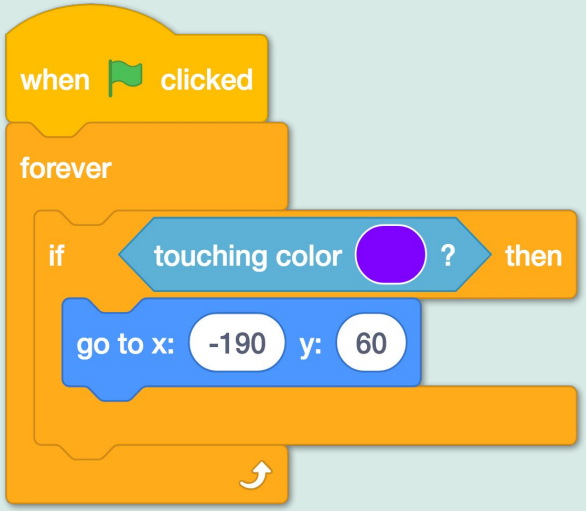


- **What do you think this code does?**
  (Answer: it moves the sprite 10 units (or steps, or pixels) to the right, or on the x plane - in Scratch the screen is 480 steps wide and 360 steps high, and the coordinate of the centre of the screen is 0,0.)
- **If this was all the code in a project, what do you think would happens if you pressed your left arrow key?**
  (Answer: nothing yet. The key hasn't been coded to do anything.)
- **What would you change if you wanted the sprite to move further each time you press the right arrow key?**
  (Answer: change 10 steps to 20 steps and see what happens.)
- **What two changes would you need to make to this code if you wanted the sprite to move to the left?**
  (Answer: change the first block to refer to clicking the left arrow, and change the unit in 'change x by' to -10, to move to the left)

Step 3: Investigating the code (continued)

Take a look at the code on the left. Play the maze again.

- **What happens when the sprite touches the maze wall?**
  (Answer: it returns to the top left corner of the screen.)
- **What would happen if you change the colour of the maze walls to green?**
  (Answer: you would no longer return to the start when you touch a wall, and could keep moving through walls as you are not touching the colour purple.)
- **Move the** `if` **block out from the** `forever` **block and reattach it to the** `when clicked` **block. What happens now?**
  (Answer: the sprite can travel through the walls. Why? Because we only check if we are touching the wall once, when the green flag is clicked. Adding forever to the code means that we are always checking whether the sprite is touching purple, instead of just once.)



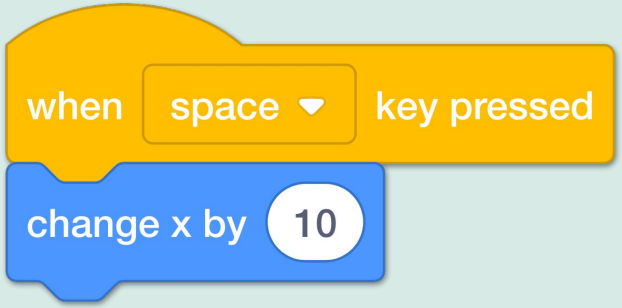
Step 4: Starting the project with inputs

The first step is to choose a sprite to travel through the maze, and control it the sprite's movement with arrow keys.

Each time we interact with our project by using keystrokes or mouse-clicks we're providing an **input** to our project. An input is data or information put into a digital system to activate or modify a process - in this case we'll modify how our sprite moves.

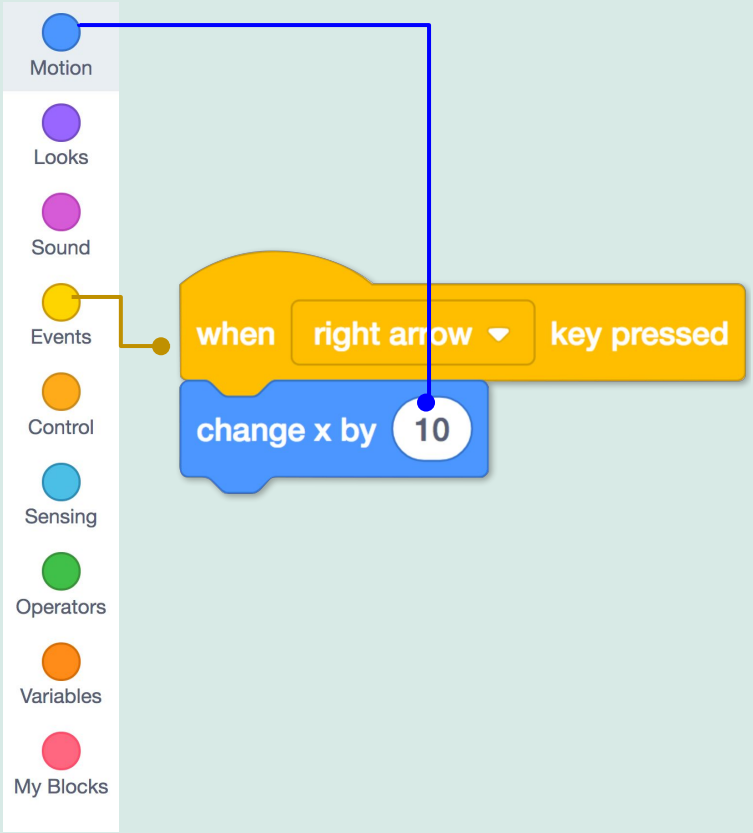
Step by step instructions:

- Choose a new sprite
- In our project the up, down, left and right arrow keys control our sprite's movement.
- Start by pulling out the when space key pressed block from the **events** drawer, and change it to say 'When right arrow key pressed'.
- From the motion blocks, pull out the change x by 10 block.
- Combine the blocks as shown.
- Test your code by pressing the right arrow key and see what happens.



Step 4: Starting the project with inputs(continued)

- The next key to code is the up arrow. There are two differences between the instruction to move right and the instruction to move up - the key pressed (up arrow) and this time as we are moving up and down instead of left or right, we change **y** by 10.
- Drag a new **when space key pressed** block from the **events** drawer
- Drag out a **change y by 10** block from the **motion** drawer and connect it.
- Students can then go ahead and create two more sets of code for the remaining two arrow keys applying their knowledge. The final code for the arrow keys is shown below.



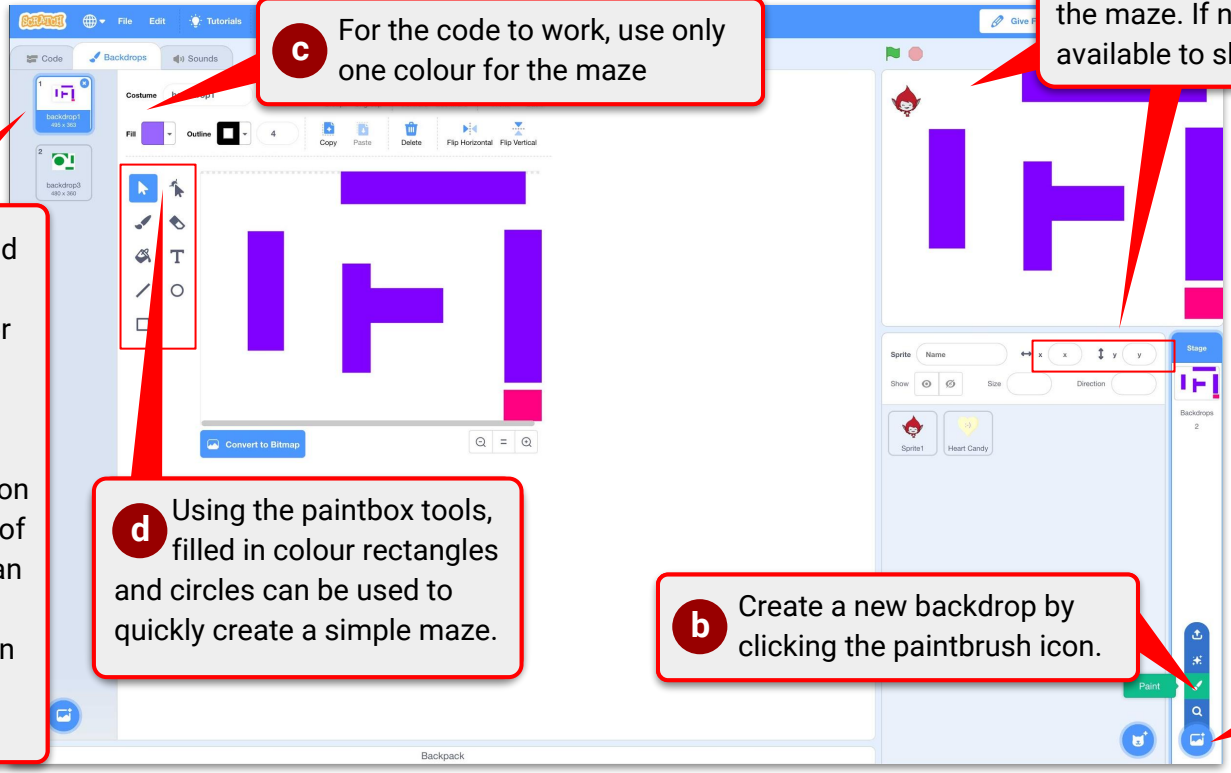
Checkpoint

The sprite is controlled by 4 arrow keys and moves around the screen.

This is a digital solution with an algorithm involving user input (ACTDIP011)

Step 5: Draw the maze

Now it's time to draw a maze using the paint tools in Scratch.



a Select the backdrop tab

b Create a new backdrop by clicking the paintbrush icon.

c For the code to work, use only one colour for the maze

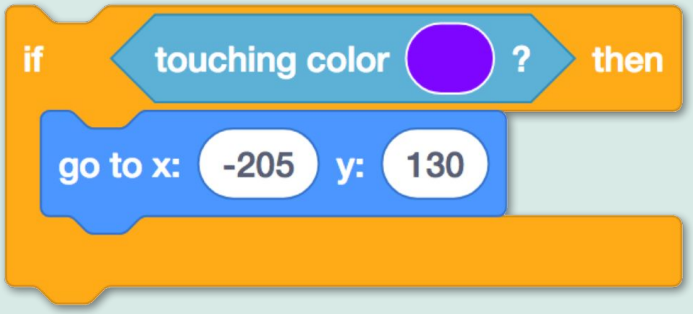
d Using the paintbox tools, filled in colour rectangles and circles can be used to quickly create a simple maze.

e Students can build A maze like the ones in the example, or can use interesting shapes to create obstacles - it's up to them. Put a time limit on creating the first level of the maze - students can always come back to their maze and work on the details later.

f Ensure that the sprite has enough space to get around the maze. If not, the resize tool is available to shrink or grow a sprite.

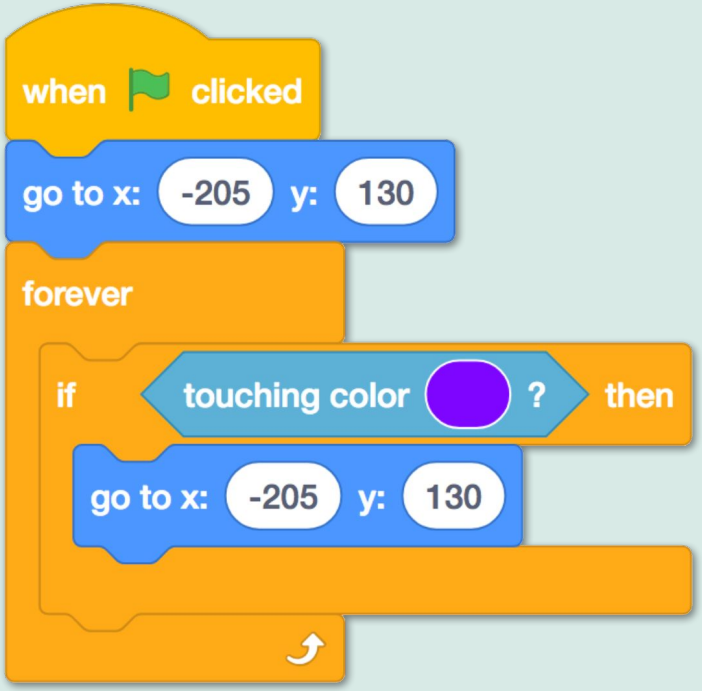
Step 6: Interact with the maze

- The maze game doesn't work yet. Brainstorm with students what the problem is.
  (Answer: the sprite travels through the maze walls).
- Currently the sprite can move through walls freely. In a real maze the walls are solid, and you have to find your way to the end by avoiding the walls.
- To fix this we're going start making decisions in our game based on whether a certain thing is true or false. This is called a conditional statement, or branching. In Scratch, we can ask 'if something is true' then do the next step. (If it's not true, the computer will skip to your next instruction).
- In this case, we'll check if our sprite is touching the colour purple. If it is, then we need to write some code to send the sprite back to the start of the maze using an x and y coordinate. If it's not touching the colour purple, then this instruction is ignored and the sprite will continue to move.



Step 6: Interact with the maze (continued)

- This code doesn't do anything yet as it's not connected to an event block. Since we want to check from the start of the game whether the sprite is touching the colour purple, we use a `when green flag is clicked` block.
- It's important that we check all the time whether or not we're touching the colour purple, instead of just once. To do this, we put the code above into a `forever` block.
- Finally, to make the maze work well we instruct our sprite to start each game in the same position on stage, using a `go to` block.
- Here is the final code.



Checkpoint

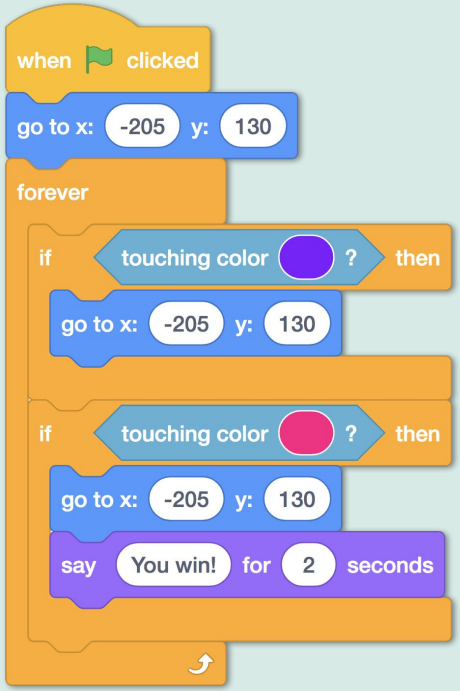
Your sprite is controlled by 4 arrow keys and moves around the screen.
It can not pass through the walls of the maze.

This is a simple digital solution involving branching (decisions) and user input (ACTDIP011)

Step 7: Add a start point and an endpoint

Now that the maze is more challenging it's time to think about what the goal of our maze is.

- Create an endpoint on your backdrop - a shape in a different colour. In our example we have added a pink rectangle to the backdrop.
- To check if the sprite has reached the end point, use the same technique as previously - if touching a colour (pink) then do something.
- Students can choose what happens to the sprite when it reaches the end. Options include sound, speech, colour effects, or returning to the start point. Encourage them to explore the looks and sound code drawers to find interesting combinations.



Checkpoint

Congratulations! You have a working maze.

The following steps increase the complexity of the maze - complete them as time permits.

Step 8: Add variables

Adding variables is a great way to add extra challenge to our game.

A variable is a place to save information in your project.

A variable can change when something happens.

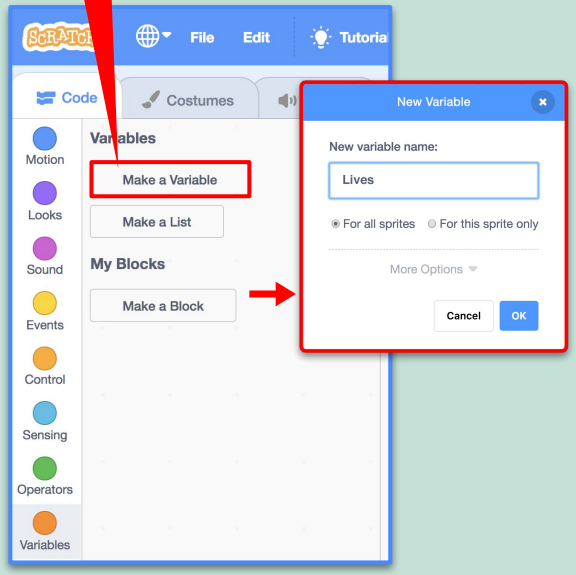
Discuss with students how lives could work in this project:

- **How many lives should the player start with?**
- **When should a player lose a life?**
- **What happens when a player has no lives left?**

The code on the next slide is an example of adding to our maze with a variable called ‘lives’, where the player starts the game with 3 lives, loses a life each time the sprite touches a wall, and ends the game when there are less than zero lives.

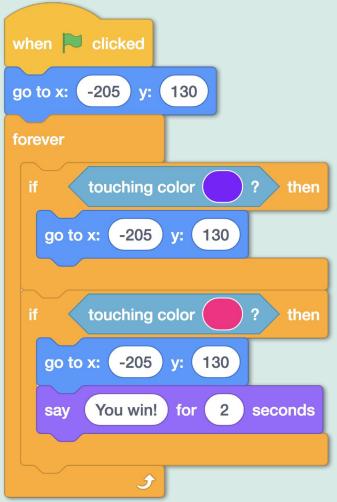
To start, follow the instructions on the right.

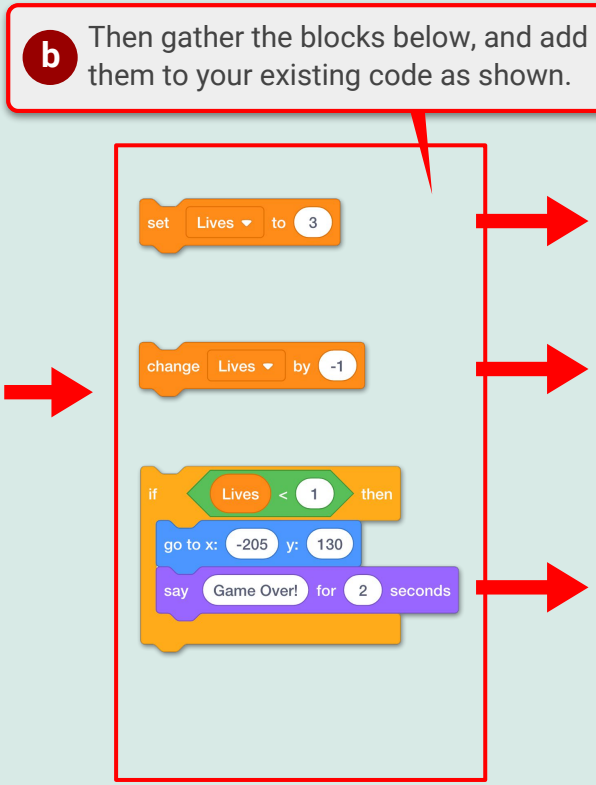
a Head to the **variables** drawer, select **make a variable**, and call it **Lives**.This will create blocks for your project to set and change how many lives you have

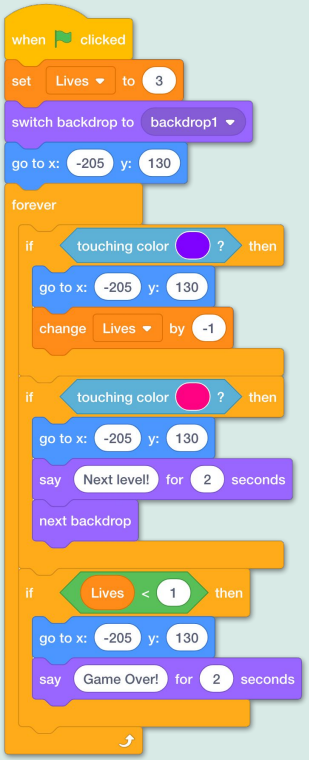


Step 8: Add variables

b Then gather the blocks below, and add them to your existing code as shown.







Step 9: Create tokens

Let's add an extra element to our maze to make it more interesting: tokens to collect. Adding tokens allows us to explore the concepts of cloning, and looping.

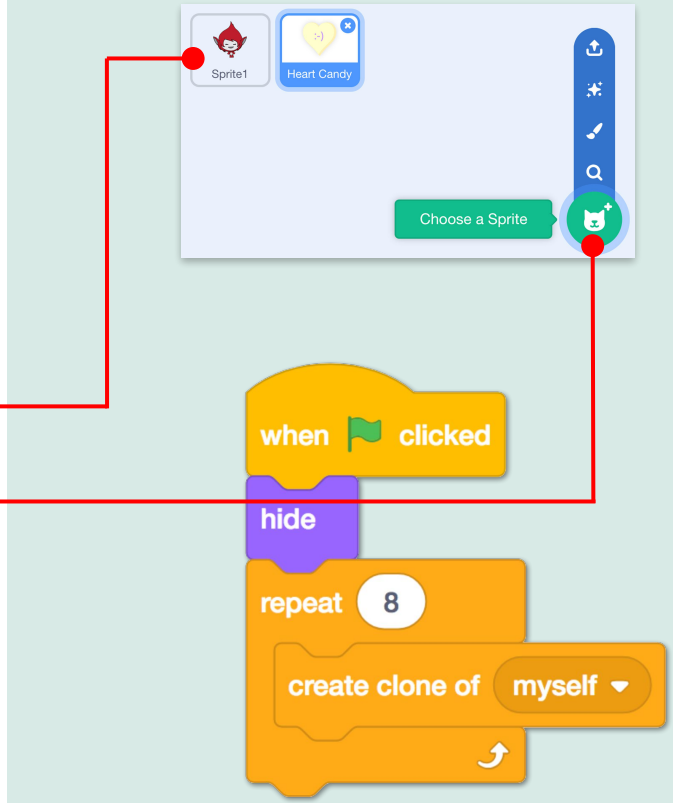
Our goal is to place eight hearts randomly around the screen, and then earn additional lives when the player collects them.

- Choose a new sprite - in the example given it's a heart candy

Note: once you have a second sprite it's important to select which sprite you want the code to apply to: you can check: the sprite being coded is highlighted blue on the menu.

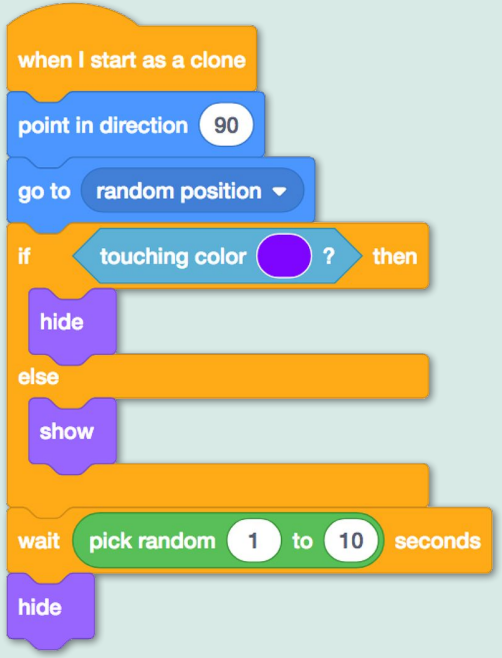
Remember, if you want to create a new sprite click here.

- Until the sprites are spread around the screen we want them to be hidden.
- Rather than making eight new sprites students can use cloning in this project. Cloning means that one sprite is copied, and code can apply to all the copies of that sprite.
- A repeat block sets the number of clones created - in our example there are 8 clones.



Step 10: Send tokens to random positions

- A **when I start as a clone** block is very useful **event** block if you want all of your clones to to react to situations independently. For example, if we want a heart to hide when we touch it, we don't want **all** the hearts to hide, just the one that is touched.
- A 'go to random position' block places copies of the sprite across the screen, however some of these will be on the solid maze walls, so can't be reached.
- We can use the 'if touching colour purple' idea again here to hide any hearts that have been placed on the walls. This means each game there will be some number between 0 and 8 hearts which show and can be collected. Notice that this time instead of using 'if' we use 'if, then, else' - this means that we will either hide or show the heart depending on whether it touches the maze walls.
- A wait block sets a time limit for how long the block will be visible. Because this code applies to each clone, each clone will appear for a different random amount of time between 1 and 10 seconds.
- Finally, the cloned sprite hides so it's no longer able to be collected.



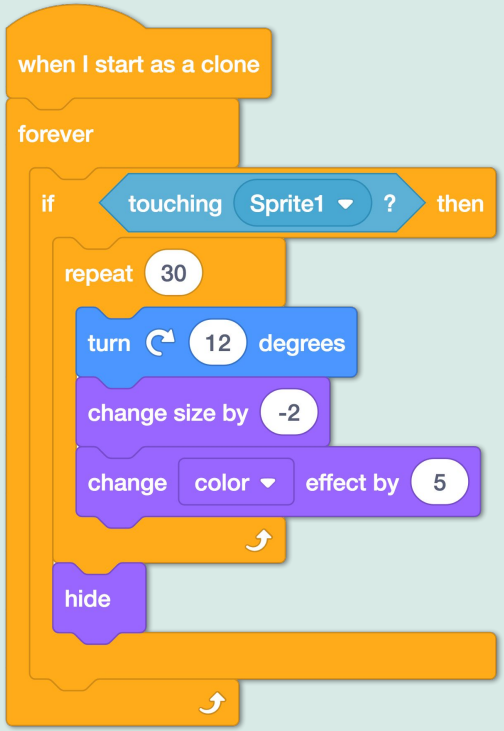
If you are using **Scratch 2**, use the 'go to mouse-pointer' block and click the triangle to choose go to random position.

Step 11: Interactions between sprites and tokens

The next step is to decide what happens when the sprite navigating the maze touches a heart.

In the example shown there are some graphical effects (changing size, colour, and rotating). There are many options for students to explore at this point. Importantly, using a repeat _ times block is an example of iterating, where an instruction is executed multiple times.

If students wish to use variables, they can add code at this point to earn extra lives by collecting hearts.



Step 12: Make it your own

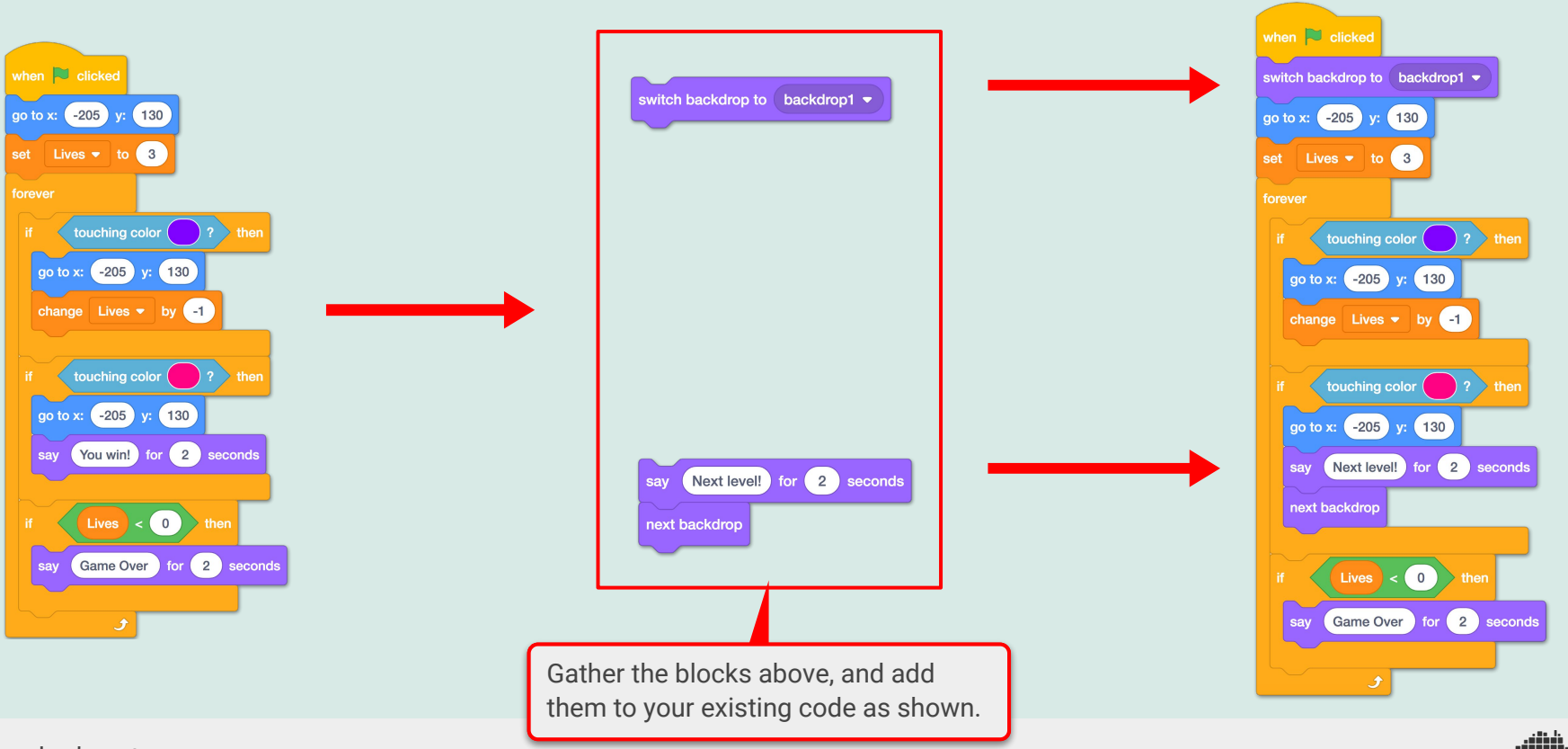
Now that the maze is working, take some time to consider with students what to do next.

Some things to try are:

- Create a second level so that when the player reaches the end point on level 1, there is a second maze (background) to navigate
- Make your second sprite (the heart) move and takes lives from the player if it touches the player
- Add another variable as health points and use a third colour on the background - when you touch it you get health points
- Use a timer to make the maze more challenging.

On the next page is an example of adding extra code to allow for more backdrops (levels) in the maze).

Step 12: Make it your own



Discussion

Reflection

Discuss with students:

- Is there anything in the maze that they would like to change or improve?
- Was there anything they found really tricky?
- Are there parts of this project they would use again in different ways?
- Encourage the students to play other students' games, and provide feedback, what did they like? Was there anything they didn't understand or felt could be improved?

Tying it back to the curriculum

Curriculum content description

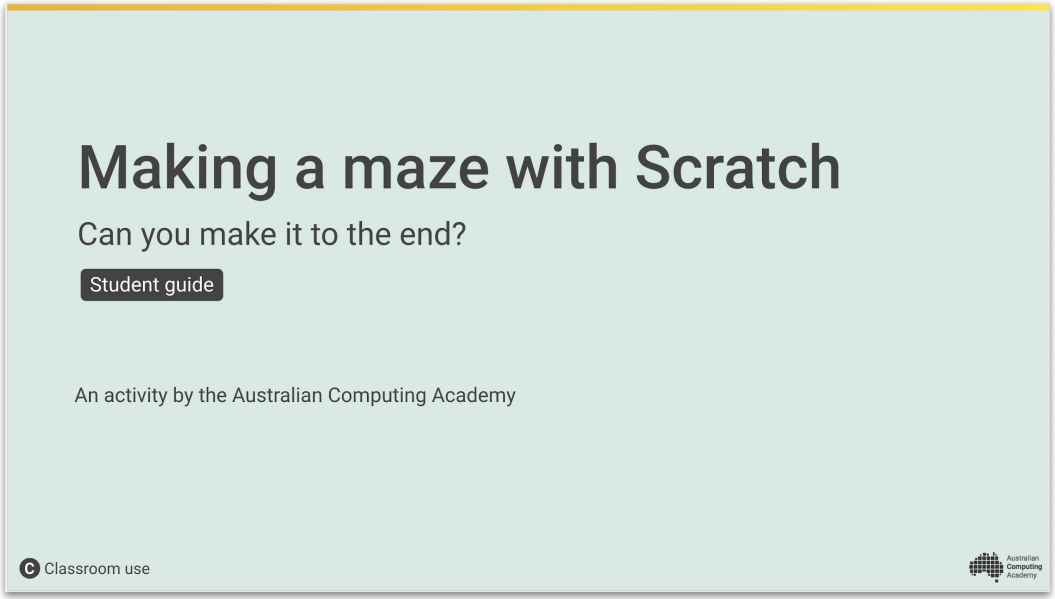
Define simple problems, and describe and follow a sequence of steps and decisions (algorithms) needed to solve them (ACTDIP010)

Implement simple digital solutions as visual programs with algorithms involving branching (decisions) and user input (ACTDIP011)

Plan, create and communicate ideas and information independently and with others, applying agreed ethical and social protocols (ACTDIP013)

Appendix

Link to student guide



Link to activity on ACA website

If you prefer students to work through the challenge at their own pace, our student guide guides students through the challenge using simplified language and without the discussion activities or curriculum links.

Student guide: cmp.ac/DTMazeStudent

Thank You!

We hope you've enjoyed exploring Scratch with the Australian Computing Academy!

You can stay in touch and hear about our new resources as we publish them by:

- Signing up to our newsletter on our website at www.aca.edu.au
- Liking our Facebook page
- Following us on Twitter
- Following us on Instagram
- Call us: 02 8627 8686

Try one of our online Digital Technologies Challenges at:
https://aca.edu.au/projects/dt-challenges/

Creative commons